iceMASTER-68HC11 Supplement

The complete *iceMASTER-68HC11* Manual is undergoing final editing and will be available very soon. Until then, you can use the information provided here to augment the information in the *iceMASTER-8051* manual and the online **Help** system.

Description of Files

The following files are supplied on the distribution diskettes for *iceMASTER-68HC11*:

ICE.EXE — The *iceMASTER-68HC11* Host Software program.

$CLR1$ — File containing default video display color and highlighting values suitable for color CGA/EGA/VGA monitors.

$CLR2$ — File containing default video display color and highlighting values suitable for color EGA/VGA monitors.

$CLRM$ — File containing default video display "color" and highlighting values suitable for monochrome monitors.

MF_GEN.EXE — Utility program used to generate a $MODEL file to allow use of a particular emulator probe card. The emulator host software (ICE.EXE) reads the $MODEL file during its initialization sequence.

> The $MODEL file defines all the specific properties of the chip in the probe card (e.g., the 68HC11A8). Such items include register definitions, memory sizes, etc. The $MODEL file also contains the text for all the online Help Topics.

During installation of the Probe Card Distribution Diskette, MF_GEN will be invoked automatically to create the $MODEL file for the probe card you purchased. MF_GEN may be reinvoked manually, if ever necessary, to regenerate the $MODEL file or to generate a different $MODEL file (for a different probe card).

MODELS.DAT — Data file read by the MF_GEN utility program. This file contains the information necessary to generate a $MODEL file for each available probe card.

MFG.EXE — This program is called only from MF_GEN. MFG should never be invoked directly at the DOS command line prompt.

Utility Program Files

HC11BOOT.AOM — This program allows you to emulate the 68HC11's Special Bootstrap operating mode with the *iceMASTER-68HC11* emulator. Detailed instructions for using HC11BOOT.AOM can be found in the assembly language source file (HC11BOOT.S07) or the assembler-generated listing file (HC11BOOT.LST; see page 9 for further details). All the files associated with the HC11BOOT program are:

HC11BOOT.AOM	Loadable, Intel absolute object module format, with symbolic information
HC11BOOT.BAT	Batch file to generate HC11BOOT from scratch
HC11BOOT.HEX	Loadable, standard hex format

HC11BOOT.LST	Assembler output listing
HC11BOOT.MAP	Linker memory map
HC11BOOT.S07	Assembly language source
HC11BOOT.S19	Loadable, Motorola S-Record format

Simple Assembly Language Demonstration Program

DEMO_H11.AOM	Loadable, Intel absolute object module format, with symbolic information
DEMO_H11.BAT	Batch file to generate DEMO_H11 from scratch
DEMO_H11.HEX	Loadable, standard hex format
DEMO_H11.LST	Assembler output listing
DEMO_H11.MAP	Linker memory map
DEMO_H11.S07	Assembly language source
DEMO_H11.S19	Loadable, Motorola S-Record format

Simple IAR/Archimedes C Language Demonstration Program

H_CSTART.LST	Assembler-generated listing file for H_CSTART.S07
H_CSTART.S07	Assembly Language source module (C runtime library startup module)
H_DEMO.AOM	Loadable, absolute object module format, (with debug information)
H_DEMO.H	C language *#include* file
H_DEMO.MAP	Linker-generated map of H_DEMO.AOM
H_HLMAIN.C	Main program (function 'main()'; C source)
H_HLMAIN.LST	Compiler-generated listing file for H_HLMAIN.C
H_INNER.C	Function 'innerloop()' (C source)
H_INNER.LST	Compiler-generated listing file for H_INNER.C
H_MAKE.BAT	Batch file to generate H_DEMO.AOM from scratch
H_MAKE.LCL	Librarian control file for generating H_DEMO.AOM
H_MAKE.XCL	Linker control file for generating H_DEMO.AOM
H_WASTE.C	Function 'wastetime()' (C source)
H_WASTE.LST	Compiler-generated listing file for H_WASTE.C

Register Bit Names

To further enhance the *iceMASTER-68HC11* emulator, we have supplied special symbols in the $MODEL file for the bit names in those registers which have special functions assigned to different bits in the register. These symbols are the standard names preceded by an at-sign (@). These symbols are special because encoded in their internal addresses are both the byte offset of the register into the Register Block and the mask to denote the bit within the register. When browsing or perusing the Register Window (*Configure* | *Windows* | *Goto* command), a box will pop-up over, for example, SCCR2,

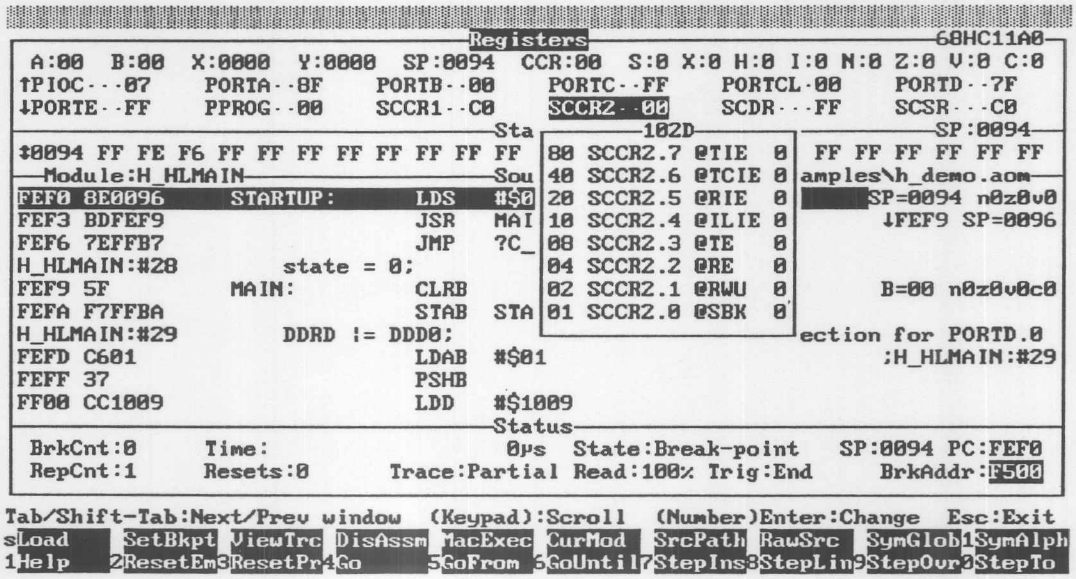

Figure 1

showing all the bits defined in that register (e.g., @RE). To toggle a bit's value, all you have to do is type the name of the bit. We preceded these names with the at-sign (@) so that these predefined symbols would not conflict with any symbolic information in your program.

iceMASTER-68HC11 always displays the individual bits of the CCR (Condition Code Register) in the top line of the Register Window. The individual bits in the CCR can be referenced symbolically using their standard names (e.g., N).

INIT Register Configuration

The 68HC11's INIT Register specifies the starting addresses of the on-chip RAM and Register Blocks. By default, the RAM block begins at 0 and the Register Block begins at 4K ($1000). If your application program requires different starting addresses for either the RAM or the Register Blocks, you can use the *Configure | Emulator | INIT Register* command to pop up a menu allowing you to set up the emulator to use a different starting location for either one, or both, of these on-chip blocks. In addition, your

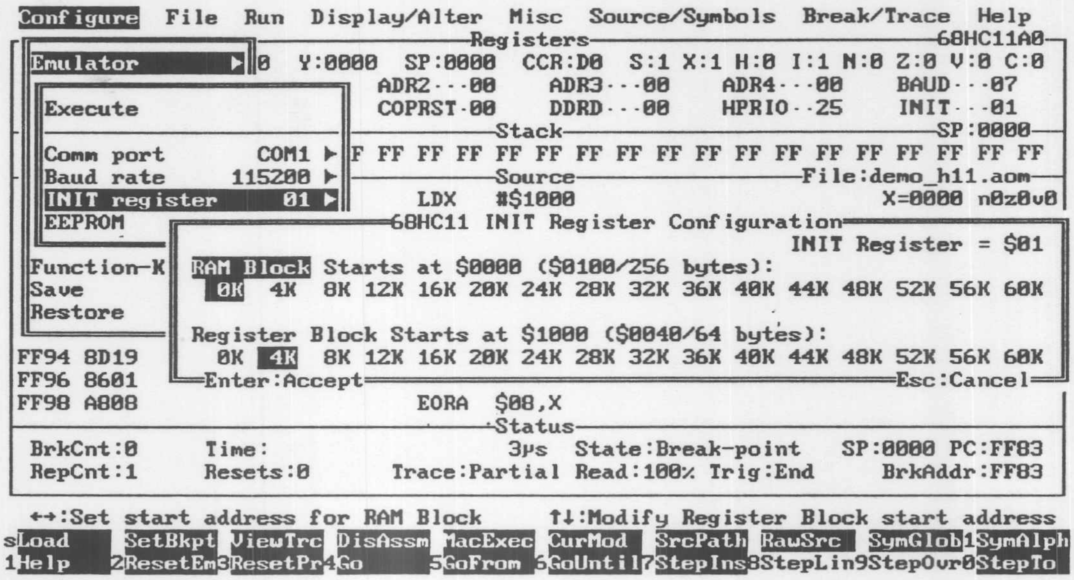

Figure 2

application program is responsible for setting the INIT register to the same value (if you start emulation using any of the emulation reset commands: *Run | Reset | Normal | Emulator*, *Run | Reset | Normal | Target*, *Run | Reset | Special | Emulator* or *Run | Reset | Special | Target*).

Each time emulation breaks, the host software and emulator firmware perform several validity checks to ensure that the INIT register does indeed contain the value specified here.

To actually effect the change to the INIT register, you must select the *Configure | Emulator | Execute* command. As part of the sequence of steps initiated by the *Configure | Emulator | Execute* command, the following code fragment is executed, starting from a Normal Mode reset, in the 68HC11 processor in the probe card:

```
86xx      LDAA   #init_value    value specified in menu
B7103D    STAA   $103D          default location of INIT
```

This is done primarily so that any Windows into the on-chip RAM (e.g., the Stack Window) or the Register Block (e.g., the Register Window) are painted initially with the current, correct values before you actually load a program and begin emulation.

The *Configure|Emulator|INIT Register* pop-up menu also shows the location and size of any other on-chip memory resource which may be, or have become, enabled (visible). This information is displayed at the bottom of the menu, as shown in Figure 3. Note that the menu in Figure 3 is shown for

```
Configure  File  Run  Display/Alter  Misc  Source/Symbols  Break/Trace  Help
 Emulator      ►0  Y:0000  SP:0096  CCR:00   S:0 X:0 H:0 I:0 N:0 Z:0 V:0 C:0
                       CONFIG·0F     COPRST·00     DDRC···00     DDRD···00
 Execute               OC1D···00     OC1M···00     OPTION·10     PACNT··00
                                  Stack                                SP:0096
 Comm port      COM1 ► F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
 Baud rate    115200 ►            Source          File:c:\examples\h_demo.aom
 INIT register    01 ►     LDS    #$0096                         SP=0000 n0z0v0
                       68HC11 INIT Register Configuration
Options                                                 INIT Register = $01
Function-K   RAM Block Starts at $0000 ($0200/512 bytes):
Save          0K  4K  8K 12K 16K 20K 24K 28K 32K 36K 40K 44K 48K 52K 56K 60K
Restore
             Register Block Starts at $1000 ($0040/64 bytes):
FEFD C601     0K  4K  8K 12K 16K 20K 24K 28K 32K 36K 40K 44K 48K 52K 56K 60K
FEFF 37
FF00 CC1009  CONFIG:  EEPROM        Block Starts at $B600 ($0200/512 bytes)
             HPRIO:   Bootstrap ROM Block Starts at $BF40 ($00C0/192 bytes)
 BrkCnt:0    CONFIG:  Code ROM      Block Starts at $D000 ($3000/12288 bytes)
 RepCnt:1      Enter:Accept                                      Esc:Cancel
 ↔:Set start address for RAM Block      ↑↓:Modify Register Block start address
sLoad     SetBkpt  ViewTrc  DisAssm  MacExec  CurMod   SrcPath  RawSrc   SymGlob1SymAlph
1Help    2ResetEm3ResetPr4Go       5GoFrom  6GoUntil7StepIns8StepLin9StepOvr0StepTo
```

Figure 3

illustrative purposes only. You should not operate the *iceMASTER-68HC11* with either the on-chip Bootstrap ROM enabled or the on-chip ROM enabled.

iceMASTER-68HC11 Emulation Environment Requirements

HPRIO.MDA = 1

Normal Expanded or Special Test mode in effect. The 68HC11 processor in the probe card must physically be operated in either the Normal Expanded mode or the Special Test mode. The *iceMASTER-68HC11* emulator supports the Normal modes (Expanded or Single-Chip) via two different probe cards: a probe card for Expanded mode and a probe card for Single-Chip mode. Either of these probe card types (Expanded or Single-Chip) can also be operated in the Special Test mode. The Special Bootstrap mode is supported via the supplied utility program HC11BOOT.AOM; commentary at the beginning of HC11BOOT.S07 (source) and HC11BOOT.LST (listing) describes how to use the *iceMASTER-68HC11* emulator to execute in the Special Bootstrap mode environment (see page 9 for further details).

CONFIG.ROMON = 0

On-chip program ROM disabled. You cannot execute in the on-chip ROM. If you should inadvertently set the ROMON bit in the CONFIG register, the host software will detect this situation when you select the *Configure|Emulator|Execute* command and attempt to recover. The recovery action consists of rewriting the CONFIG cell/register, with the ROMON bit turned off, and then resetting the 68HC11 part. This is the only time that the emulator automatically writes to the CONFIG cell. If you desire to make any other changes to the CONFIG cell/register, you can make them directly using either the *Display/Alter|Var/Reg* command, or the *Configure|Windows|Goto* command and position into the Register Window — see page 8.

HPRIO.RBOOT = 0

On-chip Bootstrap ROM disabled. You cannot execute in the on-chip Bootstrap ROM. However, see the utility program HC11BOOT (page 9), which copies the Bootstrap ROM code from the on-chip Bootstrap ROM into emulation code memory, whence the Bootstrap code can then be executed in Special Test mode.

HPRIO.IRV = 0

Internal Read Visibility disabled. We recommend that you run with IRV disabled.

Miscellaneous

Mapping

The host software will force any enabled (visible) on-chip memory resource (e.g., on-chip RAM, Register Block, on-chip EEPROM) to be mapped to the emulator. Such areas cannot be mapped to memory in your target system. After each emulation cycle, the emulator firmware and host software check to see if the status of any on-chip memory resource has changed (enabled vs. disabled). If so, the emulator will adjust the mapping automatically to account for the change.

Note: Even though the on-chip memory area is reported as being mapped to the emulator base, the memory area actually referenced during emulation is the on-chip memory resource itself.

Break-Points

If a **CBREAK: Code Break-Point** is set on an instruction, emulation breaks (stops) before that instruction executes. The same holds true for Trace-On-Points and Trace-Off-Points points.

A **WBREAK: Write Access Break-Point** at a particular location causes emulation to break (stop) after execution of an instruction writing to (storing into) that location. In such a case, the memory location is updated (i.e., the write is not suppressed). In addition, if the program's execution flow happens to pass through that location, emulation breaks before fetching the opcode from that location. Write Access Break-Points are available only in the Simple Break window.

A **PBREAK: Write Protect Break-Point** on a particular memory location causes emulation to break after execution of an instruction which attempts to write to (store into) that memory location. In such a case, the memory location is not updated (i.e., the write is suppressed). Write Protect Break-Points can only be set for a memory location/range mapped to the emulator base, not for a memory location/range mapped to the target system. Additionally, Write Protect Break-Points are available only in the Simple Break window.

You can execute code in either the on-chip RAM or the on-chip EEPROM. However, you cannot set Code or Write Protect break-points (using the *Break/Trace|Set* menu) in any of the enabled on-chip memory areas. These include the on-chip RAM, on-chip Register Block and on-chip EEPROM. If the processor is executing in an on-chip memory area (RAM or EEPROM) when a break in emulation occurs (e.g., by pressing the **Break** button on the emulator base or by pressing the **Esc** key), the software will warn you that such a condition has occurred and that the 68HC11 processor in the probe card has been reset. In such cases, you can use the Trace Buffer to find out where the program was executing when emulation stopped.

The following table summarizes which type of break-/trace-points can be used in the various memory areas:

Break-/Trace-Point Type	Break-/Trace-Point Availability				
	On-Chip Memories			64K Emulation Memory	64K Target Memory
	RAM	Registers	EEPROM		
CBREAK Code Break-Point	No	No	No	Yes	Yes
PBREAK Write Protect Break-Point	No	No	No	Yes	No
WBREAK Write Access Break-Point	Yes	Yes	Yes	Yes	Yes
TRON Trace-On-Point	Yes	Yes	Yes	Yes	Yes
TROFF Trace-Off-Point	Yes	Yes	Yes	Yes	Yes

Stack

If the Stack Pointer (SP Register) is pointing into the on-chip RAM before emulation begins, the next unused byte at the top of the stack will be changed (written) upon restarting emulation. This is a side effect of needing to restore the A Register without affecting any of the bits in the Condition Code Register (CCR).

If the SP register is pointing into any other memory area (e.g., off-chip RAM), no memory contents will be changed when emulation is restarted.

File Loading

You use the *File | Load* command to load a file into the emulator, initializing memory with the image of your program. This can be either the emulator's memory or memory in your target system, or both, depending on the current mapping (*Configure | (Mapping)Code Memory* command).

If the file contains bytes which are destined for the on-chip RAM, the on-chip RAM will be initialized with those bytes.

In addition, if the on-chip EEPROM is enabled, any bytes in the file destined for the on-chip EEPROM will be written there using the standard "erase-before-write" method, one byte at a time. However, due to optimizations employed during file loading, we recommend that, if your program file contains initialization for EEPROM, you first use the *Display/Alter | Code | Fill* command to fill all of EEPROM with $FF bytes before actually loading the file.

On-Chip Memories

When a value is written to an on-chip memory (e.g., on-chip RAM or Register Block), the corresponding location in the 64K memory in the emulator base unit is also written with the same value.

Writing EEPROM

Whenever the *iceMASTER-68HC11* writes to on-chip EEPROM memory, it pops up a box showing which locations in EEPROM are being written. This is done because writes to an EEPROM location take considerably longer than writes to other memories, due to the "erase-before-write" technique used and the need to delay 10 milliseconds after each erase and write operation.

The only time *iceMASTER-68HC11* writes to EEPROM memory is when the on-chip EEPROM is enabled and you explicitly request the write to take place (e.g., by using the *Display/Alter | Code | Fill* command or by loading a file which initializes all or part of the EEPROM).

Writing the CONFIG Cell/Register

You can change the value in the CONFIG cell/register, subject to the normal operating restrictions of the 68HC11 processor in the probe card. That is, you must first put the processor into Special Test Mode in order to be able to write to the CONFIG cell (an EEPROM location in most parts). In addition, if the processor has a BPROT register, you must at least set the PTCON bit in the BPROT register to zero before attempting to update the CONFIG cell. Finally, you will not see any immediate confirmation that the new value was actually written to the CONFIG cell; you must cause the processor to go through a reset in order to have the value in the CONFIG cell copied into the CONFIG register.

Disassemblies

In disassemblies, several opcode mnemonics have aliases (alternative mnemonic names). In such cases, the alternative name will be shown as a comment next to the disassembled instruction:

Mnemonic	Alias
ASLD	LSLD
BCC	BHS
BCS	BLO
ASLA	LSLA
ASLB	LSLB
ASL	LSL

You can use either form shown above when patching in new instructions using the single-line assembler (*Display/Alter|Asm/Dasm|Assemble* command).

Additionally, the *Display/Alter|Asm/Dasm|Disassemble* command disassembles words (byte pairs) in the current Interrupt Vector area as

```
ivect <address>
```

The current Interrupt Vector area is determined by the current operating mode of the 68HC11 processor in the probe card. The Special Modes Interrupt Vectors are located at $BFC0 through $BFFF. The Normal Modes Interrupt Vectors are located at $FFC0 through $FFFF.

Status Window

The top right hand border of the Status Window will show "Mode:Special" whenever the 68HC11 processor in the probe card is in Special Test mode (HPRIO.MDA = 1) at a break-point.

View Trace

In the *Break/Trace|View Trace* menu, when the display mode is either **Code** or **Mixed**, a small box may pop up at the currently highlighted instruction in the Trace Buffer. This box contains information extracted from the trace frames for the execution of that instruction. This information consists of values of locations/registers manipulated by that instruction. The top border of the box may show "(updated value)" to indicate that the value shown in the box is the value after the instruction executed:

View Trace Pop-Up Box Content	Applicable Instructions
`mem[xxxx]=xx`	{many}
`mem[xxxx]=xxxx`	{many}
`mem[xxxx]=xx; result=xx`	{many}
`SP=xxxx`	TSX,TSY
`CCR=xx,B=xx,A=xx,X=xxxx,Y=xxxx,Ret=xxxx`	RTI,SWI

TEST Instruction

If the TEST instruction (opcode value $00) is executed during emulation in Special Test Mode, a box will pop up informing you that the processor in the probe card is inactive. If you press **Esc**, the host software will ask you whether or not you want to reset the 68HC11 processor to break (stop) emulation.

ACTIVE Light

When the green **ACTIVE** light on the emulator base is on, it indicates that the $\overline{\text{LIR}}$ and AS signals in the 68HC11 are toggling properly.

XIRQ

When emulation stops, XIRQ is disabled. When emulation begins/resumes, XIRQ is enabled.

Reading Ports

The origin of Port values displayed in the Register Window depends on the configuration of each port pin. If the pin is configured as an output pin, the value displayed is that from the port register. If the pin is configured as an input pin, the value displayed is that from the port pin.

Special Bootstrap Mode

The HC11BOOT.AOM utility program allows you to use the *iceMASTER-68HC11* emulator to execute in the Special Bootstrap mode environment. The listing for HC11BOOT appears on the following pages. The commentary box at the beginning of the listing provides detailed instructions for using HC11BOOT.

Processor Used The processor actually used in each probe card is as follows:

Probe Card Type	Processors Supported	Processor Used in Probe Card
68HC11 Expanded Mode	68HC11A0 68HC11A1	68HC11A1
	68HC11D0	68HC11D0
	68HC11E0 68HC11E1	68HC11E1
	68HC811E2	68HC811E2
	68HC11F1	68HC11F1
68HC11 Single-Chip Mode	68HC11A7 68HC11A8	68HC11A1
	68HC11D3 68HC711D3	68HC11D0
	68HC11E8 68HC11E9 68HC711E9	68HC11E1
	68HC811E2	68HC811E2

All probe cards are shipped from the factory with the on-chip EEPROM memory disabled (CONFIG.EEON = 0) and with the COP (Computer Operating Properly) WDT (Watchdog Timer) disabled (CONFIG.NOCOP = 1). Additionally, for those parts having a moveable EEPROM (e.g., 68HC811E2 and 68HC11F1), the CONFIG register is set up so that if the EEPROM is enabled, the EEPROM initially will not be at the high end of memory (i.e., not covering the Normal Modes interrupt/reset vectors).

Jumper Blocks All *iceMASTER-68HC11* probe cards, whether Expanded mode or Single-Chip mode, have only one user-selectable jumper block:

XTAL – Clock Source

The function of the **XTAL Jumper** is to select the source of the clock (oscillator) for the 68HC11:

PC: Probe card's crystal is used.

TAR: Target system supplies crystal or external clock.

This is a double jumper to ensure correct configuration. The center post is the common post.

68HC11E*x* Probe Cards

We use the 68HC11E1 chip in the probe cards which support the following devices:

68HC11E0

68HC11E1

68HC11E8

68HC11E9

68HC711E9

The 68HC11E1 physically has an EEPROM/CONFIG Block Protect Register – BPROT. BPROT is initialized out of Reset (every time) to prevent writing to the EEPROM memory and to the CONFIG cell. The 68HC11E0 and 68HC11E8 chips do not have a BPROT Register (i.e., they do not have this secondary level of protection against writing to some EEPROM cell).

If you are developing a 68HC11E0 or 68HC11E8 application using the *iceMASTER-68HC11* emulator, you must account for the fact that the BPROT Register physically exists in the chip in the probe card. The simplest solution is to add code which clears (zeroes-out) the BPROT Register immediately after a Reset. If you wish, you can even leave this BPROT-clearing code in when you go into production with your application. Writing to a non-existent register in the Register Block is harmless in most cases.

68HC11D*x* Probe Cards

When using a probe card supporting the 68HC11D0, 68HC11D3 or 68HC711D3, you must explicitly clear the ROMON bit in the CONFIG register after every reset. The default state of the ROMON bit in the CONFIG register following a reset is 1 (ROM enabled).

Single-Chip Mode Probe Cards

All *iceMASTER-68HC11* Single-Chip mode probe cards use either the MC68HC24 or MC68HC27 Port Replacement Unit to recreate the single-chip mode functions of Ports B and C. The resultant timing characteristics of the probe card reflect the internal timing in the MC68HC24/MC68HC27 device with respect to STRA, STRB, PORTB and PORTC. Refer to the appropriate data book (MC68HC24 or MC68HC27) for complete details.

The *iceMASTER-68HC11* host software forces the registers at $x002-$x007 to appear to be mapped to your target system. This is done only to support the operational characteristics of the MC68HC24 or MC68HC27 Port Replacement Unit in the probe card.

Finally, for all Single-Chip mode probe cards, it is safest to initialize the INIT register explicitly after every reset. The reason is that even though the INIT register in the 68HC11 part is time-protected in Normal Expanded mode, the mirror INIT register maintained in the MC68HC24/MC68HC27 PRU part is a write-once register. That is, this mirror INIT register can be written at any time after each reset, but it can only be written once after each reset. Normally, this would present no problem. However, if the first attempt to write to the INIT register occurs later than the first 64 cycles following a reset, the new value will be written to the mirror INIT in the PRU but not to the INIT in the 68HC11. Conversely, you should write to the INIT register only once following a reset. Again, the mirror INIT in the PRU can be written only once following a reset; all other writes to this mirror INIT, even if they occur within 64 cycles of the reset, will be ignored (unlike the 68HC11 part itself).